Izgradnja aplikacije za pretprocesiranje teksta

Rački, Anela

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
SMJER INFORMATIKA – ISTRAŽIVAČKI
Ak. god. 2023/2024

Anela Rački

Izgradnja aplikacije za pretprocesiranje teksta

Diplomski rad

Mentor: doc. dr. sc. Ivan Dunđer

Zagreb, lipanj 2024.

Izjava o akademskoj čestitosti

Izjavljujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Hvala mentoru na savjetima, preporukama i pomoći pri izradi ovog rada i aplikacije.

Hvala roditeljima i obitelji na podršci i motivaciji tokom studija.

Hvala svima koji su bili uz mene na ovom putu i pomagali mi čak i kad sam mislila da sve mogu sama.

Sadržaj

1. Uvod

Tekstualni podaci rijetko stižu u upotrebljivom obliku; zapravo, može se reći da podaci nikada nisu besprijekorni. Tekstovi prikupljeni iz različitih izvora i okruženja stvarnog svijeta, sadrže brojne pogreške i usvajaju različite formate. Kako bi prikupljeni tekstualni podaci bili upotrebljiv za analizu i pravilnu interpretaciju bitno ih je adekvatno očistiti (Salton, 1988: 6). Pretprocesiranje teksta, u obradi prirodnog jezika, je praksa čišćenja tekstualnih podataka, uklanjanje suvišnih informacija i konvertiranje podataka u format prihvatljiv za daljnju obradu.

Robustan sustav za pretprocesiranje teksta uvijek je bitan dio bilo koje aplikacije za obradu prirodnog jezika i analizu teksta (Sakar, 2016). Kvalitetno pretprocesiranje teksta (tzv. predobrada) je bitno jer tekstualne komponente dobivene pretprocesiranjem, čine osnovne gradivne blokove koji se koriste u daljnjim fazama analize i obrade koje obavljaju složenije zadatke, uključujući obrasce učenja i „izvlačenje“ informacija (Sakar, 2016). Pretprocesiranje teksta je ključno za uklanjanje bilo kakve pristranosti ili neželjenih efekata u analizi. Pravilno rukovanje s nepotpunim podacima ili nejednolikostima u korpusu teksta može spriječiti iskrivljavanje rezultata analize ili modela strojnog učenja. Stoga je popularna izreka „trash input, trash output“ (hrv. smeće ulazno, smeće izlazno) ovdje relevantna jer će neadekvatno obrađeni podaci, u konačnici rezultirati neželjenim i neispravnim rezultatima (Sakar, 2016). Kvalitetna predobrada značajna je i u područjima u kojima se modeli treniraju iz velikih količina teksta, jer loše pretprocesiranje može dovesti do nastanka šuma u modelima, smanjenja njihove performanse ili, u najgorem slučaju, do generiranja neispravnih zaključaka.

Obrada teksta također pomaže u čišćenju i standardizaciji teksta, što pomaže u analitičkim sustavima, poput povećanja točnosti klasifikatora (Dean, 2014: 175). Dodatno, važno je napomenuti da pretprocesiranje teksta nije samo korak koji se provodi jednokratno prije analize, već je često iterativan proces koji zahtijeva kontinuiranu pažnju i optimizaciju. Različiti izvori tekstualnih podataka mogu zahtijevati različite tehnike pretprocesiranja, ovisno o specifičnostima teksta i ciljevima analize.

Unatoč svojoj važnosti, pretprocesiranje teksta smatra se toliko ustaljenim korakom obrade prirodnog jezika da se o njemu rijetko i piše. Primjerice, Hickman et al. (2020) ističu kako velika većina studija zatvorenog vokabulara za rudarenje teksta ne izvješćuje o provođenju bilo kakve prethodne obrade, unatoč preporukama da se svi tekstovi prethodno obrade kako bi se pospješila preciznost mjerenja. Osim u rudarenju teksta, pretprocesiranje je važan korak u svim

modelima strojnog učenja, obrade i analize jezika. U svijetu sve veće količine dostupnih tekstualnih podataka, efikasno i pouzdano pretprocesiranje postaje ključno za dobivanje vrijednih uvida i informacija iz teksta.

2. Konceptualna i empirijska razmatranja pretprocesiranja teksta

S pojavom ogromnih količina nestrukturiranih podataka i uspjeha s tehnikama strojnog učenja i statističke analize, nije prošlo dugo prije nego što je analiza teksta počela privlačiti veliku pozornost (Sakar, 2016: 49). Mogućnost izvlačenja korisnih informacija i smislenih uvida iz velike količine nestrukturiranih i neobrađenih tekstualnih podataka danas je od velike važnosti (Denny i Spirling, 2018). Za razliku od uobičajenih analitičkih metoda, analiza teksta donosi nove izazove te zahtijeva prilagodbu metoda predobrade (Denny i Spirling, 2018). Tekstualni podaci predstavljaju izazov u procesiranju i konvertiranju iz nestrukturiranih podataka u podatke prikladne za modeliranje (Dean, 2014: 175).

Neizostavan dio lingvističkih istraživanja i statističkih analiza prirodnog jezika su tekstualni korpusi, a također se koriste i kao podaci za izradu NLP alata. Kao i drugi tekstualni podaci, tekstualni korpusi u izvornom *raw* formatu nisu dobro formatirani i standardizirani (Sakar, 2016: 40). Tekstovi su vrlo nestrukturirani i rijetko slijede bilo koji specifičan obrazac – poput vremenskih podataka ili strukturiranih atributa u relacijskim bazama podataka (Salton, 1988: 6). Stoga standardne statističke metode nisu od pomoći kada se odmah primjenjuju na nestrukturirane tekstualne podatke (Sakar, 2016: 50).

Svaka kvantitativna studija koja koristi tekst kao podatak zahtijeva prvotno pretprocesiranje kako bi unose za analizu bilo jednostavnije interpretirati i njima manipulirati (Denny i Spirling, 2018). Predobrada podataka (eng. *data preprocessing*) bitan je korak u mnogim poljima analize podataka, izgradnje modela strojnog učenja, obrade prirodnog jezika i sl. (Sakar, 2016: 50).

Problemi obrade podataka, prema Saltonu (1988: 6) jednostavniji su i bolje razumljivi za tekstualne podatke nego za druge vrste informacija. „Tekst prirodnog jezika može se predstaviti kao jednodimenzionalni niz znakova, dok su govor i slike inherentno dvodimenzionalni.“ (Salton, 1988: 6).

Aplikacije za obradu teksta kao što su Microsoft Word, Google Docs, Dropbox Paper, LibreOffice Writter, iCloud Pages i dr. koje se danas rutinski koriste, nude širok raspon tekstualnih operacija koje se uglavnom temelje na prepoznavanju riječi i pojedinačnih znakova unutar riječi. Ove operacije uključuju metode automatskog uređivanja i oblikovanja teksta, otkrivanje i ispravljanje pravopisnih pogrešaka, pretraživanje i pronalaženje pojedinih riječi unutar teksta, te usporedbu riječi koje se pojavljuju u tekstovima s pohranjenim podacima iz

rječnika zbog određivanja svojstava riječi kao što su sintaktičke funkcije ili prevedeni oblici riječi.

Pretprocesiranje teksta uključuje korištenje različitih tehnika za pretvaranje *raw* teksta u dobro definirane sekvence jezičnih komponenti koje imaju standardnu strukturu i zapis (Sakar, 2016: 50). Ovaj korak u obradi teksta uključuje korištenje tehnika obrade prirodnog jezika (eng. *Natural Language Processing*, *NLP*), pronalaženja informacija (eng. *Information retrieval*) i metoda strojnog učenja (eng. *Machine Learning*, *ML*) za pretvaranje nestrukturiranih tekstualnih podataka u strukturirane oblike te izvođenje uzoraka i uvida iz tih podataka koji bi bili od pomoći krajnjem korisniku (Sakar, 2016: 107).

2.1. Rudarenje i analiza tekstualnih podataka

S porastom Big Data, podataka koji se generiraju, prikupljaju i pohranjuju u velikim bazama i skladištima podataka, porasla je i potreba za primjenom tehnika analize podataka (Dean, 2014: 175). Stoga je potrebno pripremiti podatke prije analize, kako bi se mogli izvući kvalitetni uvidi iz velikih količina podataka. Preporučeni koraci u pripremi podataka su identificiranje problema kojeg se pokušava riješiti i identificiranje potencijalnih izvora podataka (Dean, 2014: 175). Primjena ovih koraka omogućuje organizacijama da iskoriste potencijal svojih podataka i donesu informirane odluke.

Analiza podataka je proces sustavnog prikupljanja, čišćenja, transformacije, opisivanja, modeliranja i tumačenja podataka, uglavnom korištenjem statističkih metoda (Eldridge, 2024). Covington (2016: 73) analizu podataka, definira kao proces prikupljanja sirovih podataka (eng. *raw data*) iz različitih izvora i njihovo konvertiranje u smislene informacije koje se mogu koristiti za donošenje odluka (eng. *decision making*). Tukey (navodi Covington, 2016:73) analizu podataka definira kao:

> „procedure za analizu podataka, tehnike za interpretaciju rezultata tih procedura, načini na koje se planira prikupljanje podataka kako bi njihova analiza bila jednostavnija, preciznija ili točnija, te svi strojevi i rezultati (matematičkih) statistika koji se primjenjuju na analizu podataka“.

Analiza podataka važan je dio znanstvenog istraživanja i poslovanja, gdje je posljednjih godina porasla potražnja za donošenjem odluka na temelju podataka (Eldridge, 2024).

Analiza teksta, također poznata kao rudarenje teksta (eng. *text mining*), je metodologija koja se koristi za ekstrakciju informacija iz tekstualnih podataka (Sakar, 2016: 49) i pronalaženje novih, prethodno neidentificiranih informacija iz različitih pisanih izvora (Vijayarani et al., 2015). Dean (2014:178) ističe rudarenje teksta kao disciplinu koja kombinira računalnu analizu i rudarenje podataka kako bi nestrukturirane, tj. tekstualne podatke koristili zajedno sa strukturiranim podacima u svrhu istraživanja, pronalaženja podataka (eng. *data retrieval*), prediktivnog modeliranja i klasifikacije. Analiza teksta sastoji se od skupa tehnika strojnog učenja, lingvističkih i statističkih tehnika koje se koriste za modeliranje i ekstrakciju informacija iz teksta prvenstveno za potrebe analize, uključujući poslovnu inteligenciju, istraživačku, deskriptivnu i prediktivnu analizu (Sakar, 2016: 50). Bitan korak u analizi teksta je pronalaženje informacija (eng. *Information retrieval*) koji se odnosi na alate za pretraživanje weba i dokumenata (eng. *web crawling*, *file crawling*), ekstrakciju teksta (eng. *text extraction*) i indeksiranje (eng. *indexing*) (Dean, 2014:176).

Ovaj proces pronalaženja ili izvlačenja korisnih informacija iz tekstualnih podataka s ciljem otkrivanja znanja iz nestrukturiranih tekstova, također je poznat kao rudarenje tekstualnih podataka (eng. *Text Data Mining*, *TDM*) i otkrivanje znanja u tekstualnim bazama podataka (eng. *Knowledge Discovery in Textual Databases*, *KDTB*) (Vijayarani et al., 2015). Oba ova područja imaju važnu ulogu u analizi i upravljanju velikim količinama tekstualnih podataka koje se generiraju na internetu, u poslovnim dokumentima, u znanstvenim istraživanjima i drugdje. KDTB igra sve značajniju ulogu u novim aplikacijama, kao što je razumijevanje teksta (eng. *Text Understanding*) (Vijayarani et al., 2015). Korištenje TDM i KDTB omogućuje organizacijama i istraživačima da iskoriste bogatstvo informacija koje se krije u tekstualnim podacima pretvarajući ih u korisno znanje.

Proces rudarenja teksta isti je kao rudarenje podataka, osim što su alati za rudarenje podataka dizajnirani za rukovanje strukturiranim podacima, dok alati za rudarenje teksta mogu rukovati nestrukturiranim ili polustrukturiranim skupovima podataka kao što su HTML datoteke, e-pošta ili dokumenti s punim tekstom (Vijayarani et al., 2015), zbog čega je provođenje analize teksta ponekad zahtjevniji proces od uobičajene statističke analize ili strojnog učenja (Sakar, 2016: 49). Prije primjene bilo kojeg algoritma ili tehnike učenja, tekst je potrebno pretprocesirati, tj. nestrukturirane tekstualne podatke pretvoriti u format prihvatljiv za te algoritme.

2.2. Obrada teksta i prirodnog jezika

Prikupljeni tekstualni podaci iz različitih izvora i različitih formata nisu nužno odmah upotrebljivi za obradu ili analizu, zbog čega je ključno njihovo čišćenje, odnosno pretprocesiranje. Čišćenje teksta je važan korak u obradi prirodnog jezika (NLP) i strojnom učenju, zbog pripreme neobrađenih tekstualnih podataka za analizu, modeliranje i vizualizaciju. Alati za čišćenje podataka, također mogu poboljšati kvalitetu i dosljednost tekstualnih podataka, što može imati značajan utjecaj na izvedbu i pouzdanost NLP modela.

Obrada prirodnog jezika je interdisciplinarno područje računalnih znanosti te umjetne inteligencije s korijenima u računalnoj lingvistici (Sakar, 2016: 46), koje se bavi praktičnim aspektima primjene računala za potrebe izvršavanja zadataka koji uključuju ljudski, prirodni jezik, odnosno načine na koje se računala mogu koristiti za razumijevanje i manipuliranje tekstom ili govorom prirodnog jezika (Chowdhury, 2003). U tu svrhu obrada prirodnog jezika često posuđuje ideje iz teorijske lingvistike kako bi se moglo uz pomoć tehnologije točno „izvući“ informacije i uvide sadržane u dokumentima, kao i kategorizirati i organizirati te same dokumente (Sakar, 2016: 46).

NLP tehnike omogućuju računalnu obradu i razumijevanje prirodnog/ljudskog jezika i daljnju upotrebu za pružanje korisnih rezultata (Sakar, 2016:46). Primjene ovih tehnika uključuju brojna znanstvena i istraživačka polja, kao što su strojno prevođenje, obrada teksta na prirodnom jeziku i sažimanje, korisnička sučelja, višejezično i međujezično pronalaženje informacija (eng. *Cross-language information retrieval*, *CLIR*), prepoznavanje govora, umjetna inteligencija, ekspertni sustavi itd. (Chowdhury, 2003).

U obradi prirodnog jezika, pretprocesiranje teksta je praksa čišćenja i pripreme tekstualnih podataka. Prema Hickman et al. (2020) preoblikovanje teksta prije njegove analize uključuje četiri koraka:

1. uklanjanje sadržaja koji je irelevantan (pr. uklanjanje neabecednih znakova i zaustavnih riječi),
2. aglomeracija semantički povezanih izraza kako bi se smanjila oskudnost podataka i povećala prediktivna moć (pr. dekapitalizacija slova, ispravljanje pravopisnih pogrešaka, proširenje kontrakcija i kratica te korjenovanje i lematizacija),
3. utvrđivanje jedinica (riječi i izraza) za korištenje (tj. označavanje, eng. *tagging*),
4. povećanje količine semantičkih informacija koje se dohvaćaju (tj. rukovanje negacijom) (Hickman et al., 2020).

Primjenom različitih tehnika za pretprocesiranje, tekst se pretvara u vektor riječi, tj. numerički niz specifičnih vrijednosti za svaku pojedinu riječ (Sakar, 2016: 50).

Cilj pretprocesiranja je pojednostaviti unose analize tako da ne utječu negativno na interpretabilnost ili suštinske zaključke modela (Denny i Spirling, 2018). Rezultat pretprocesiranja, ističu Denny i Spirling (2018), su jednostavniji podaci, no bez većeg gubitka informacija, stoga ne iznenađuje da se takvi savjeti, koji uključuju prednosti operacija kao što je dekapitalizacija, korjenovanje riječi i uklanjanje vrlo čestih riječi, mogu pronaći u publikacijama koje tematiziraju obradu prirodnog jezika i pronalaženje informacija.

3. Istraživanje

U sljedećim poglavljima rada opisan je tijek izgradnje aplikacije za pretprocesiranje teksta. Prikazana je metodologija izgradnje aplikacije, osnovne značajke razvojnog okruženja Python i modula korištenih u procesu izrade. Opisan je proces pretprocesiranja teksta te su prikazane tehnike pretprocesiranja i način njihove primjene u aplikaciji. Također su prikazane implementacije metoda analize kolokacija i konkordancija. Na kraju je provedeno testiranje i evaluacija aplikacije. Rezultati su uspoređeni s drugim, sličnim aplikacijama.

3.1. Metodologija

U svrhu stvaranja funkcionalnog alata za obradu teksta, metodologija izrade aplikacije za pretprocesiranje teksta slijedi nekoliko koraka. Na početku su definirani zahtjevi aplikacije. Cilj je bio razviti alat koji može učitati tekstualne datoteke različitih formata (pdf, docx, html, txt) i izvršiti pretprocesiranje teksta, uključujući tokenizaciju, uklanjanje HTML oznaka, specijalnih znakova, brojeva, te stop riječi na hrvatskom jeziku. Kako bi alat bio upotpunjen i dobra polazišna točka za druga istraživanja u njega su uključene i analiza kolokacija i konkordancija. Analiza kolokacija fokusira se na identifikaciju čestih kombinacija riječi radi razumijevanja jezičnih uzoraka, dok analiza konkordancija pruža detaljan pregled svakog pojavljivanja određene riječi ili fraze u kontekstu radi dubljeg razumijevanja njihove upotrebe.

Nakon definiranja zahtjeva, dizajnirano je korisničko sučelje koje omogućuje korisniku dodavanje datoteka, pretprocesiranje teksta, analizu konkordancija i kolokacija te pohranu rezultata. Korisničko sučelje uključuje gumb za dodavanje datoteka, gumb za pokretanje pretprocesiranja i prikaz učitanih datoteka, gumb za pokretanje analize konkordancija i ispis kolokacija te pregled rezultata.

Implementirane funkcionalnosti aplikacije uključuju:

- učitavanje teksta iz datoteka različitih formata (pdf, docx, html, txt),
- pretprocesiranje teksta uključujući tokenizaciju, uklanjanje HTML oznaka, specijalnih znakova, brojeva i stop riječi,
- pohranu pretprocesiranog teksta u novu datoteku,
- analizu kolokacija, odnosno pronalazak i ocjenjivanje kolokacija temeljenih na PMI mjeri,

- analizu konkordancija tražene riječi u okolini.

Nakon implementacije, provjerena je ispravnost aplikacije provođenjem testiranja. Testirane su funkcionalnosti poput učitavanja različitih formata datoteka, ispravnosti pretprocesiranja teksta, analize i točnosti spremanja rezultata. Tijekom testiranja identificirane su i otklonjene greške koje su se pojavile. To uključuje debagiranje koda i izvršavanje ispravaka kako bi se osigurao ispravan rad aplikacije.

3.2. Razvojno okruženje

Za izgradnju aplikacije za pretprocesiranje teksta odabran je programski jezik Python jer nudi širok raspon biblioteka i alata posebno dizajniranih za obradu i analizu teksta. Od svog početka u kasnim 1980-ima, Python je dizajniran da bude proširiv jezik visoke razine s velikom standardnom bibliotekom i jednostavnom, izražajnom sintaksom (Dean, 2014: 49). Može se koristiti interaktivno ili skriptno, a često se koristi za skriptiranje, numeričku i tekstualnu analizu, razvoj web aplikacija i dr. (Dean, 2014: 49).

Python zajednica razvila je veliko i aktivno znanstveno računalstvo što Python čini jednim od najvažnijih programskih jezika za podatkovnu znanost, strojno učenje i opći razvoj softvera u akademskoj zajednici i industriji (McKinney, 2018:2). U kombinaciji s ukupnom snagom Pythona za softverski inženjering opće namjene, McKinney (2018:2) ga ističe kao izvrsnu opciju za izradu aplikacija, a posebno važan je za analizu podataka, interaktivno računalstvo i vizualizaciju podataka.

Moduli se u Pythonu promatraju kao biblioteke kodova koje sadrže skupove funkcija koje se koriste u raznim aplikacijama (Dovedan Han, 2021: 223). To su zbirke unaprijed napisanih programskih kodova koji pružaju širok raspon funkcionalnosti za obavljanje specifičnih zadataka. Moduli u pravilu sadrže funkcije, klase i konstante za višekratnu upotrebu koje pomažu programerima da pojednostave proces kodiranja i izbjegnu dupliciranje programskog koda. Standardna biblioteka programa sadrži preko dvije stotine modula čiji broj varira od inačice do inačice (Dovedan Han, 2021: 223). Python biblioteke pokrivaju različite domene, uključujući podatkovnu znanost, web razvoj, strojno učenje, obradu prirodnog jezika i dr. U rukovanju s tekstualnim podacima uobičajene su biblioteke nltk, gensim, TextBlob i spaCy za obradu prirodnog jezika, pronalaženje informacija i analitiku teksta (Sakar, 2016). Za zadatke analize podataka uglavnom se koriste biblioteke pandas i scikit-learn, dok se nltk i re koriste

za zadatke obrade i pretprocesiranja teksta (McKinney, 2018:2). Matematički i statistički modeli mogu se implementirati korištenjem biblioteka Scipy i Numpy, koje imaju snažnu podršku za linearnu algebru, numeričku analizu i statističke operacije (Dean, 2014: 49). Također su moguće i primjene standardnih algoritama strojnog učenja za rješavanje problema povezanih s analizom teksta (Sakar, 2016).

U aplikaciji za pretprocesiranje teksta korišteni su sljedeći moduli:

1. *fitz*, *Document* i *BeautifulSoup* – manipulacija datoteka,
2. *nltk* – obrada prirodnog jezika (tokenizacija teksta, uklanjanje stop riječi, lematizacija, korjenovanje riječi i dr.),
3. *re* – obrada teksta pomoću regularnih izraza (prepoznavanje i zamjena uzoraka u tekstu),
4. *tkinter* – izrada korisničkog sučelja.

MODUL FITZ

Modul *fitz* iz PyMuPDF biblioteke koristi se za izdvajanje teksta iz PDF datoteka. PyMuPDF je Python vezni sloj za MuPDF[1] biblioteku koja se koristi za rad s PDF dokumentima. To je biblioteka visokih performansi za ekstrakciju podataka, analizu, konverziju i manipulaciju PDF (i drugim) dokumentima (PyPi, bez god.).

U aplikaciji se ova biblioteka koristi za obradu PDF datoteka kako bi se izdvojio tekst iz svake stranice, kako je prikazano u programskom kodu 1:

```
def izdvoji_tekst_iz_pdfa(pdf_putanja):
    tekst = ''
    pdf_dokument = fitz.open(pdf_putanja)
    for broj_stranice in range(len(pdf_dokument)):
        stranica = pdf_dokument.load_page(broj_stranice)
        tekst += stranica.get_text()
    pdf_dokument.close()
    return tekst
```

Programski kod 1. Modul fitz – ekstrakcija sadržaja iz PDF dokumenta

[1] MuPDF je preglednik za PDF, XPS i e-knjige. Sastoji se od softverske biblioteke, alata naredbenog retka i preglednika za različite platforme. Renderer u MuPDF-u prilagođen je visokokvalitetnoj anti-aliasing grafici. Renderira tekst s metrikom i razmacima točnim unutar frakcija piksela za najveću vjernost u reprodukciji izgleda ispisane stranice na ekranu (*MuPDF*, https://mupdf.com).

MODUL DOCUMENT

Biblioteka *docx* koristi se za čitanje, pisanje i manipulaciju Microsoft Word dokumentima u Pythonu. Klasa *Document* omogućuje čitanje sadržaja iz DOCX datoteka te se koristi za njihovu obradu i izdvajanje teksta iz svakog paragrafa (Das, 2023). Ovu biblioteku nije razvio Microsoft, već su je kao projekt otvorenog koda izradili pojedinci u Python zajednici kako bi pružili način za programsko stvaranje i manipuliranje .docx datotekama (Das, 2023). Sadržaj iz Word dokumenta se ekstrahira tako da petlja prolazi kroz dokument te vraća listu svih paragrafa koji se pohranjuju u string, kako je prikazano u programskom kodu 2:

```
def izdvoji_tekst_iz_docx(docx_putanja):
    doc = Document(docx_putanja)
    tekst = ''
    for paragraf in doc.paragraphs:
        tekst += paragraf.text
    return tekst
```

Programski kod 2. Modul Document – ekstrakcija sadržaja iz Word dokumenta

BIBLIOTEKA BEAUTIFUL SOUP

Beautiful Soup je biblioteka dizajnirana za ekstrakciju sadržaja s weba (eng. *web scraping*), raščlanjivanje i analizu HTML i XML dokumenata. Jedna od ključnih prednosti Beautiful Soupa je njegova sposobnost rukovanja loše formatiranim HTML-om, što ga čini otpornim na varijacije i nedosljednosti u strukturama web stranica. Nudi različite metode i svojstva za pretraživanje, filtriranje i navigaciju kroz HTML strukture, omogućujući korisnicima da lociraju određene elemente i učinkovito „izvuku“ željene informacije. Štoviše, Beautiful Soup se neprimjetno integrira s drugim Python bibliotekama kao što su zahtjevi za dohvaćanje web stranica i lxml za raščlanjivanje XML dokumenata, te kako je prikazano u programskom kodu 3, omogućuje jednostavno parsiranje HTML-a i izdvajanje sadržaja iz njega.

```
def izdvoji_tekst_iz_html(html_putanja):
    with open(html_putanja, 'r', encoding='utf-8') as
datoteka:
        soup = BeautifulSoup(datoteka, 'html.parser')
        tekst = soup.get_text()
        return tekst
```

Programski kod 3. Biblioteka BeautifulSoup – ekstrakcija sadržaja iz HTML dokumenta

BIBLIOTEKA NLTK

NLTK, skraćeno od *Natural Language Toolkit* je biblioteka za obradu prirodnog jezika u Pythonu. Sadrži više od 50 korpusa i leksičkih resursa te pruža potrebne alate, sučelja i metode za obradu i analizu tekstualnih podataka (Sakar, 2016). Prikladna je za izvođenje zadataka kao što su tokenizacija, korjenovanje i lematizacija riječi, anotacija/označavanje, analiza sentimenta i dr., što ju čini praktičnom za pretprocesiranje tekstualnih podataka (Bird, Klein, Loper, 2009).

NLTK je nazvan „prekrasnim alatom za poučavanje i rad u računalnoj lingvistici koristeći Python“ i „nevjerojatnom bibliotekom za igranje s prirodnim jezikom“ (Bird, Klein, Loper, 2009). Korištenje ovih alata, ističe Sakar (2016: 50), štedi mnogo truda i vremena koje bi se inače potrošilo na pisanje standardnog koda za obradu i manipuliranje tekstualnim podacima, i na taj način omogućuje razvojnim programerima i istraživačima da se više usredotoče na rješavanje stvarnih problema i potrebne logike te potrebnih algoritama.

MODUL RE

Pythonov modul *re*, skraćeno od *Regular Expressions* (hrv. regularni izrazi), alat je namijenjen za rad s tekstualnim uzorcima. „Značenje regularnih izraza je sparivanje određenih nizova (riječi), u skladu s određenim pravilima.” (Dovedan Han, 2021: 224). Modul *re*, koristi se u programima za uređivanje teksta, pretragu i manipuliranje izrazima (Dovedan Han, 2021: 224). Nudi funkcije za pretraživanje, analizu podudarnosti i rukovanje nizovima pomoću regularnih izraza, koji su nizovi znakova koji definiraju uzorak pretraživanja.

U aplikaciji za pretprocesiranje teksta se ovaj modul koristi za uklanjanje neabecednih znakova, prepoznavanje i uklanjanje URL linkova iz teksta i tokenizaciju dokumenta na paragrafe.

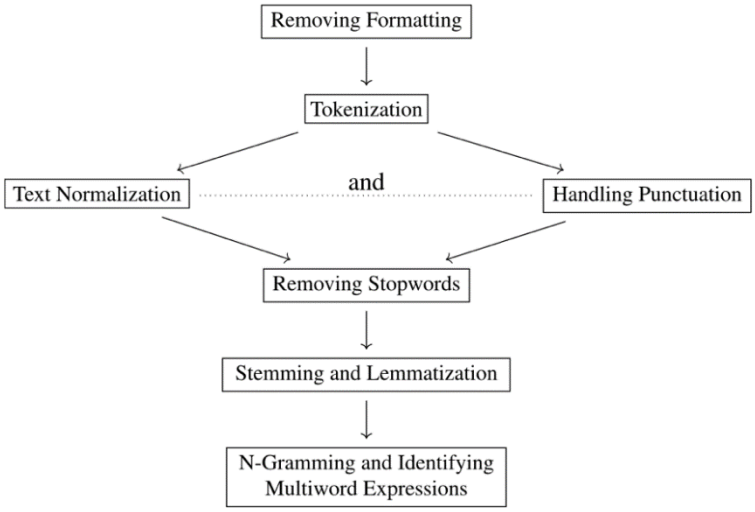
4. Pretprocesiranje i tehnike obrade tekstualnih podataka

Pretprocesiranje teksta ima temeljnu ulogu u obradi prirodnog jezika (NLP) i zadacima rudarenja teksta pripremajući neobrađene tekstualne podatke za analizu (*GeeksforGeeks*, bez god.). Uključuje tehnike koje uzimaju u obzir više semantičkih informacija tretirajući različite oblike iste riječi kao jednu jedinicu kako bi se povećala valjanost rezultata, tj. kako bi se smanjila dimenzionalnost teksta (Hickman et al., 2020). Glavni cilj pretprocesiranja teksta je dobiti ključne značajke iz tekstualnog dokumenta i poboljšati relevantnost između riječi i dokumenta te relevantnosti između riječi i klase[2] (Kadhim, 2018), te u konačnici pretvaranje tekstualnih podataka u format koji je prikladan za daljnju obradu i analizu (*GeeksforGeeks*, bez god.). Ipak, ističe Chai (2023), pretprocesiranje teksta složenije je nego što se čini, jer tekst sadrži mnoge vrste leksičkih informacija kao što su koncept, gramatika, sintaksa i morfologija. Iako gramatika nije važna za modeliranje teme ili klasifikaciju teksta, ključna je za aplikacije poput onih koji odgovaraju na pitanja ili rade sažimanje teksta (Torres-Moreno, 2014). S druge strane, složenost predobrade teksta potvrđuju Srividhya i Anitha (2010) te Kadhim (2018) izjavom da „vrijeme potrošeno na pretprocesiranje može trajati od 50% do 80% cjelokupnog procesa klasifikacije teksta“, čime jasno ukazuju na važnost pretprocesiranja teksta.

Primjene pretprocesiranja teksta su raznolike i obuhvaćaju različite NLP zadatke, uključujući analizu sentimenta, klasifikaciju teksta, modeliranje i pronalaženje informacija (*GeeksforGeeks,* bez god.). U analizi sentimenta, tehnike pretprocesiranja pomažu očistiti i standardizirati tekstualne podatke prije klasifikacije sentimenta, poboljšavajući točnost predviđanja sentimenta. Slično, u zadacima klasifikacije teksta, pretprocesiranje osigurava da su tekstualni podaci dosljedni i informativni, što dovodi do bolje izvedbe klasifikacije. Štoviše, tehnike pretprocesiranja bitne su u modeliranju tema za pripremu tekstualnih podataka za izdvajanje tema i algoritme grupiranja, olakšavajući identifikaciju tema i trendova unutar velikih tekstualnih korpusa (*GeeksforGeeks,* bez god.). Pretprocesiranje teksta nije univerzalan postupak jer različite vrste tekstualnih korpusa zahtijevaju i različite metode pretprocesiranja (Denny i Spirling, 2018).

[2] Klasa se odnosi na skupinu predmeta, pojava, pojmova, bića i dr. koji imaju jedno ili više zajedničkih svojstava; razred, kategorija. U informatici, klasa (eng. *class*) je temeljni koncept objektu orijentiranog programiranja (*klasa*. Hrvatska enciklopedija) koji sadrži opis, tj. definiciju određenog pojma. Primjerice kasa „dokument“ može sadržavati informacije o nazivu dokumenta, autoru, datumu kreiranja i/ili posljednje izmjene, broj znakova, jezik dokumenta i sl.

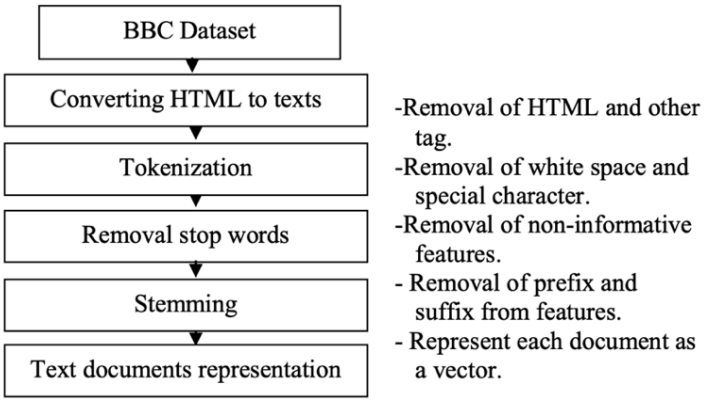
Chai (2023) objašnjava uobičajeni redoslijed primjene modula za pretprocesiranje teksta. Ovi moduli (slika 1) uključuju uklanjanje formatiranja teksta, tokenizaciju, normalizaciju teksta, rukovanje interpunkcijskim znakovima, uklanjanje zaustavnih riječi, korjenovanje i lematizaciju, popisivanje n-grama i identifikaciju višerječnih izraza (Chai, 2023).



Slika 1. Uobičajeni redoslijed primjene modula za pretprocesiranje teksta

Izvor: Chai, 2023

S druge strane Kadhim (2018) (slika 2), za prvi korak pretprocesiranja teksta određuje uklanjanje HTML i drugih oznaka nakon čega slijedi podjela teksta na manje sastavne jedinice, tj. tokene. Zatim slijedi uklanjanje neinformativnih tokena (stop riječi, brojevi i posebni znakovi) te standardizacija tokena (korjenovanje). Posljednji je korak reprezentacija dokumenta u vektorskom prostoru (Kadhim, 2018).

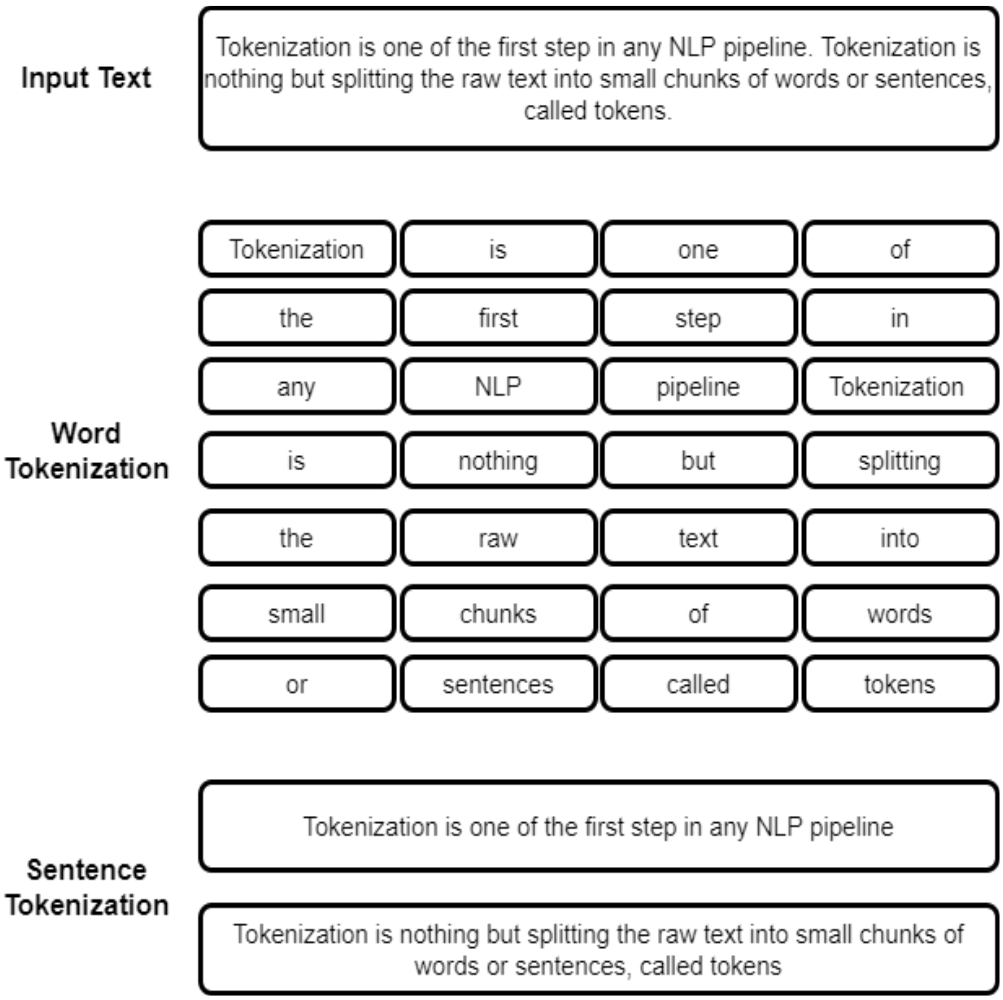


Slika 2. Koraci pretprocesiranja teksta

Izvor: Kadhim, 2018

4.1. Tokenizacija

Tokenizacija je proces rastavljanja velikih blokova neobrađenog teksta kao što su odlomci i rečenice u manje jedinice, tj. tokene, kojima se lakše upravlja (Hickman et al., 2020) (slika 3). Ovaj proces jedan je od prvih koraka svih NLP zadataka. Pretvaranje teksta u tokene je bitno kako bi stroj mogao razumjeti ljudski jezik razlučujući ga na manje dijelove koje je jednostavnije analizirati (Awan, 2023).



Slika 3. Tokenizacija riječi i rečenice

Izvor: Hickman et al., 2020

Metode tokenizacije razlikuju se ovisno o granularnosti raščlambe teksta i specifičnim zahtjevima zadatka (Awan, 2023), a najčešće se izvode tokenizacija riječi i rečenice. Također postoji i više načina izvođenja tokenizacije zadanih tekstualnih podataka.

Tokenizacija riječi posebno je učinkovita za jezike s jasnim granicama riječi kao što su hrvatski i engleski jezik (Awan, 2023). Kod tokenizacije riječi se uglavnom koristi „razmak“ kao separator (Hickman et al., 2020), a najčešće se izvodi pomoću `.split()` metode. Kod tokenizacije rečenice se kao separatori postavljaju točka, uskličnik i upitnik (Hickman et al., 2020). Tokenizacija znakova korisna je za jezike koji nemaju jasne granice riječi ili za zadatke koji zahtijevaju detaljnu analizu, kao što je ispravljanje pravopisa, jer se tekst segmentira u pojedinačne znakove (Awan, 2023). Uspostavljajući ravnotežu između tokenizacije riječi i znakova, tokenizacija podriječi rastavlja tekst u jedinice koje mogu biti veće od jednog znaka, ali manje od cijele riječi (Awan, 2023). Ovaj je pristup posebno koristan za jezike koji oblikuju značenje kombiniranjem manjih jedinica ili kada se u NLP zadacima radi s riječima izvan vokabulara (Awan, 2023). Prilikom izvođenja tokenizacije, znakovi razmaka i interpunkcije se zanemaruju i ne svrstavaju se u konačan popis tokena (Hickman et al., 2020).

U aplikaciji izrađenoj u sklopu ovog diplomskog rada, implementirana je tokenizacija rečenica i riječi kako bi se omogućila daljnja obrada i analiza. Prvotno je tekst podijeljen na paragrafe, pretpostavljajući da je svaki paragraf odvojen novim redom. Ovaj korak obavlja se pomoću `.split('\n')` metode, te rezultira listom paragrafa. Nadalje se svaki paragraf dijeli na rečenice, korištenjem `nltk.sent_tokenize()` funkcije, kako bi izlazni podatak bila lista rečenica unutar svakog paragrafa. Svaka dobivena rečenica dalje se dijeli na pojedinačne riječi. Ovaj korak obavlja se unutar petlje koja prolazi kroz sve rečenice. Korištenjem funkcije `nltk.word_tokenize()`, svaka se rečenica razbija na pojedinačne riječi koje se zatim pohranjuju u listu tokena (Programski kod 4).

```
paragrafi = sadržaj.split('\n')
pretprocesirani_tekst = ''

for paragraf in paragrafi:
     rečenice = nltk.sent_tokenize(paragraf)

     for rečenica in rečenice:
...
          tokeni = nltk.word_tokenize(rečenica)
```

Programski kod 4. Tokenizacija rečenica i riječi

Nakon što je dobiven popis riječi, mogu se koristiti statistički alati i metode kako bi se dobio bolji uvid u značenje teksta. Provođenjem tokenizacije riječi dobiva se točniji prikaz temeljnih obrazaca i trendova prisutnih u tekstualnim podacima (Awan, 2023).

4.2. Dekapitalizacija teksta

Pisanje velikih slova, osim na početku rečenice, uglavnom se koristi za identifikaciju vlastitih imenica i akronima, u suprotnom ne nosi semantičke informacije (Hickman et al., 2020). Računala različito predstavljaju velika i mala slova, tako da se ista riječ napisana velikim slovima u odnosu na istu napisanu malim slovima može računati zasebno (Hickman et al., 2020), tj. računaju se kao dvije različite riječi (npr. Pas ≠ pas). Dekapitalizacija korpusa je najčešći oblik normalizacije teksta (Chai, 2023). Proces dekapitalizacije uključuje pretvaranje svih slova u dokumentu u mala slova. Ovaj korak se vrši Pythonovom ugrađenom metodom `.lower()`, a implementirana je kako algoritam ne bi tretirao iste riječi različito. Procesom dekapitalizacije teksta smanjuje se dimenzionalnost podataka, čime se povećava statistička valjanost rezultata analize, a istovremeno se ne mijenja značenje teksta (Chai, 2023). Glavne prednosti normalizacije velikih i malih slova su dosljednost i konsolidacija varijacija riječi, zadržava se jednostavnost prilikom ekstrakcije tekstualnih značajki i smanjuje broj različitih tokena (Chai, 2023). Korištenje malih slova također se preporučuje u pronalaženju informacija jer nije vjerojatno da će se upiti za pretraživanje pisani velikim slovima podudarati sa sadržajem u korpusu, koji može biti napisan i velikim i malim slovima (Chai, 2023).

4.3. HTML oznake i URL linkovi

Uklanjanje HTML oznaka korak je pretprocesiranja teksta koji se koristi za čišćenje tekstualnih podataka iz HTML dokumenata. Tekstualni podaci dobiveni s web stranica ili drugih izvora formatiranih u HTML-u, sadrže HTML oznake koje nisu poželjne u analizi teksta ili modelima strojnog učenja. Stoga je važno ukloniti HTML oznake iz tekstualnih podataka.

HTML oznake u tekstu se uklanjaju korištenjem biblioteke *BeautifulSoup*. Kako je prikazano u programskom kodu 5, koristi se metoda `get_text()` koja izdvaja samo tekstualne podatke iz HTML oznaka.

```
rečenica = BeautifulSoup(rečenica, 'html.parser').get_text()
```

Programski kod 5. Metoda .get_text() – uklanjanje HTML oznaka

Format URL-ova može biti složen, stoga osmišljavanje regularnog izraza koji odgovara bilo kojem mogućem URL-u može biti jednako složen zadatak (Friedl, 2006: 25). U aplikaciji

izgrađenoj u sklopu ovog diplomskog rada se URL linkovi uklanjaju primjenom funkcije `re.sub()` unutar koje se definira regularni izraz koji prepoznaje URL uzorke i zamjenjuje ih praznim stringom (Programski kod 6). Ovaj izraz dizajniran je kako bi prepoznao većinu tipičnih URL-ova, uključujući http i https protokole, domene, putanje, upite i fragmente. Iako je teško odrediti mogući naziv hosta, tj. web poslužitelja i ostatka URL putanje, prema Friedl (2006: 25) treba koristiti izraz koji će prepoznati sve znakove koji su dopušteni u URL-ovima. To uključuje prepoznavanje velikih i malih slova, brojeva i posebnih znakova, a postiže se izrazom: `[a-zA-Z]|[0-9]|[$-_@.&+]` kojeg se upotpunjuje s `[!*\\(\\),]`. Ovim se izrazima prepoznaju sva slova (velika i mala), brojevi te posebni znakovi. Kako bi se prepoznale heksadecimalne vrijednosti koje se mogu pojaviti poput „%20“ za prazninu ili „%2F“ za kosu crtu („/“), izraz se dopunjuje s `(?:%[0-9a-fA-F][0-9a-fA-F])`. Na kraju se znakom „+“ označava da se prethodni dio uzorka pretraživanja može pojaviti jednom ili više puta, što znači da URL može sadržavati više znakova nakon početka.

```
  rečenica = re.sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]
|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', '', rečenica)
```

Programski kod 6. Funkcija re.sub() – uklanjanje URL linkova

4.4. Posebni znakovi

Uklanjanje neabecednih znakova uključuje uklanjanje interpunkcijskih znakova, simbola i/ili brojeva iz teksta. Posebnim, tj. specijalnim znakovima smatraju se svi znakovi koji nisu alfanumerički ili razmaci. To su interpunkcijski znakovi, simboli i kontrolni znakovi.

Ako se neabecedni znakovi zadrže u tekstu, svaka instanca n-grama pored interpunkcije (npr. na kraju rečenice) može se tretirati odvojeno od drugih instanci, stoga brojni istraživači predlažu da se uvijek uklone interpunkcijski znakovi (Banks et al., 2018). Neki posebni znakovi mogu imati posebna značenja unutar same sintakse regularnog izraza. Na primjer, znak točke („ . “) je zamjenski znak koji odgovara bilo kojem znaku u regularnom izrazu, pa ako se želi postići podudaranje s doslovnim znakom točke, potrebno ga je označiti obrnutom kosom crtom („ \“).

Ponekad ipak nije poželjno ukloniti neabecedne znakove, posebno kada treba istražiti individualne razlike poput karaktera teksta, jer je interpunkcija jedan od bitnih stilskih

elemenata teksta (Pennebaker et al., 2015). Na primjer, upotreba uskličnika povezana je s osobinama likova u tekstu (Golbeck et al., 2011), a interpunkcija je potrebna i za kvantificiranje indeksa jezične složenosti koji koriste broj riječi po rečenici (Pennebaker et al., 2015). Točke se također mogu pojaviti u ispisu vremena (npr. 18.30), adresama e-pošte (npr. korisničkoime@website.com) i dr., sto dodatno otežava izradu točnih regularnih izraza.

Kao i za uklanjanje URL-ova iz teksta, posebni se znakovi iz teksta uklanjaju upotrebom regularnih izraza i funkcije `re.sub()` kako bi se zamijenili svi znakovi koji nisu riječi s praznim nizom. Regularni izraz `r'\W+'` odgovara jednom ili više znakova koji nisu riječi u nizu. Klasa znakova `\W` odgovara bilo kojem znaku koji nije riječ, uključujući interpunkcijske znakove, simbole i razmake. Kvantifikator + znači da se prethodna klasa znakova (u ovom slučaju `\W`) treba podudarati jednom ili više puta. No, za uklanjanje svih nealfanumeričkih znakova u hrvatskom jeziku češća je upotreba modificiranog regularnog izraza `r'[^a-zšđčćž\s]+`, u kojem znak `^` unutar uglatih zagrada (`[]`) označava da se traži bilo koji znak koji nije u klasi znakova koji slijede. U ovom slučaju, klasa znakova uključuje sva slova hrvatske abecede i razmake. Ovim se regularnim izrazom učinkovito uklanjaju svi nealfanumerički znakovi iz teksta, a pritom se zadržavaju razmaci.

4.5. Zaustavne riječi

Najčešće riječi u tekstualnim dokumentima su veznici, prijedlozi i zamjenice, koje ne doprinose značenju rečenice (Vijayarani et al., 2015) te se mogu ukloniti bez ikakve promjene u značenju teksta. Takve riječi nazivaju se zaustavnim ili stop riječima, te se smatraju toliko uobičajenima u korpusu da mogu postati neinformativne zbog čega se često uklanjaju tijekom pretprocesiranja (Hickman et al., 2020). Dodatna motivacija za njihovo isključenje iz korpusa je činjenica da tekst čine težim i manje važnim za analitičare (Vijayarani et al., 2015). Primjerice, u pronalaženju informacija, zaustavne riječi su rijetko korisne, a njihovo uklanjanje smanjuje vrijeme provođenja analize (Hickman et al., 2020).

Kako bi se smanjila dimenzionalnost pojmovnog prostora (Vijayarani et al., 2015), istraživači predlažu da se zaustavne riječi uvijek uklone (Banks et al., 2018), osim u kratkim dokumentima (Kobayashi et al., 2018), ili ako se ne predviđaju individualne razlike, učestalosti pojavljivanja zaustavnih riječi i sl. (Kern et al., 2016). Zaustavne riječi su funkcijske riječi koje čine govorni

stil stoga se one ne uklanjaju u istraživanjima koja se odnose na analizu govornog ili pisanog stila (Hickman et al., 2020).

NLTK biblioteka sadrži skup zaustavnih riječi koji se može koristiti za uklanjanje zaustavnih riječi iz teksta i vraćanje popisa tokena riječi. Osim generičkih popisa zaustavnih riječi NTLK biblioteke, mogu se koristiti i popisi specifični za određenu domenu (Hickman et al., 2020).

Vijayarani et al. (2015), izdvaja četiri metode uklanjanja zaustavnih riječi iz korpusa:

1. *Klasična metoda* – temelji se na uklanjanju zaustavnih riječi dobivenih iz unaprijed sastavljenih popisa;
2. *Z-metode* (metode temeljene na Zipfovom zakonu) – tri metode kreiranja stop riječi temeljene na Zipfovom zakonu:
    a. TF-High (Term frequency-High) – uklanjanje najčešćih riječi,
    b. TF1 (Term frequency-one) – uklanjanje riječi koje se pojavljuju jednom, tj. pojedinačne riječi,
    c. IDF (Inverse Document Frequency) – uklanjanje riječi s niskom inverznom frekvencijom dokumenta;
3. *Metoda uzajamne informacije* (eng. *Mutual Information Method*, *MI*) – nadzirana metoda kojom se izračunavaju uzajamne informacije između danog pojma i klase dokumenta (npr. pozitivna ili negativna) sugerirajući koliko informacija dani pojam može reći o klasi dokumenta. Slaba uzajamna informiranost sugerira da pojam ima nisku diskriminatornu moć te bi ga stoga trebalo ukloniti;
4. *Nasumično uzorkovanje temeljeno na terminima* (eng. *Term-Based Random Sampling*, *TBRS*) – metodu predlažu Lo et al. (2005) za ručno otkrivanje stop riječi iz web dokumenata. Ovom metodom iteriraju se nasumično odabrani dijelovi podataka te se zatim rangiraju pojmovi iz svakog dijela na temelju njihovih vrijednosti koristeći Kullback-Leiblerovu[3] mjeru divergencije. Konačna lista stop riječi konstruira se uzimanjem najmanje informativnih pojmova iz svih iteriranih dijelova dokumenta.

U ovoj aplikaciji za pretprocesiranje teksta, stop riječi za hrvatski jezik definirane su u skupu `hr_stopwords`, na temelju kojih se provodi filtriranje teksta (Programski kod 7).

[3] Kullback-Leiblerova (KL) metrika divergentnosti (relativne entropije) je statističko mjerenje u teoriji informacija koje se često koristi za kvantificiranje razlika između jedne vjerojatnosti distribucije i referentne vrijednosti distribucije. KL divergencija jedna je od temeljnih metrika u teoriji informacija koja kvantificira blizinu dviju distribucija vjerojatnosti (Shlens, 2014).

Koristeći `nltk.word_tokenize()`, tekst se razdvaja na tokene na razini riječi, a zatim se filtriraju samo one riječi koje nisu u skupu stop riječi. Nakon uklanjanja stop riječi, preostali tokeni se ponovno spajaju u rečenice, odnosno tekst. Ovim se procesom iz teksta uklanjaju sve riječi koje se smatraju nevažnima ili koje ne pridonose značenju teksta, čime se poboljšava kvaliteta analize i obrade teksta.

```
hr_stopwords = set(["i", "a", "ili", "ali", "pa", "te", "u",
        "s", "na", "po", "koji", "koja", "koje", "kojima",
        "kojemu", "kojim", "kojoj", "kojeg", "bez", "od",
        "do", "iz", "kroz", "preko", "između", "ispod",
        "iznad", "nad", "pod", "kao", "jer", "budući",
        "također", "izvijestiti"])
tokeni = nltk.word_tokenize(rečenica)
filtrirani_tokeni = [token for token in tokeni if token not
in hr_stopwords]
```

Programski kod 7. Uklanjanje stop riječi

5. Analiza teksta

Analiza teksta (eng. *Text analysis*) koristi se za automatsko izdvajanje vrijednih uvida iz nestrukturiranih tekstualnih podataka. Ova se tehnika često koristi za proučavanje specifičnih jezičnih elemenata unutar teksta, poput riječi, fraza ili struktura. Provođenjem analize teksta otkrivaju se obrasci u jezičnoj upotrebi, semantičke veze između riječi te različite leksičke i gramatičke karakteristike. Jedan od izazova u obradi prirodnog jezika je dvosmislenost koja se pojavljuje u značenju riječi, morfologiji, sintaktičkim svojstvima i vezama između dijelova teksta (Šuman, 2021). Ovaj se problem rješava uzimanjem šireg konteksta oko riječi i zaključivanjem na temelju prošlih slučajeva (Šuman, 2021). Kontekst može biti rečenica ili izraz (Riahi i Sedghi, 2016).

Uz pretprocesiranje teksta, analiza kolokacija i analiza konkordancija ključni su koraci u obradi i analizi teksta. Ovi su koraci međusobno povezani i često se koriste zajedno kako bi se bolje razumjeli jezični obrasci i kontekstualni sadržaj teksta. Analiza učestalosti pojavljivanja određene riječi također je uobičajena u obradi prirodnog jezika, jer vrlo česte riječi mogu pružiti dublje uvide u određene pojave (Pavlovski i Dunđer, 2018).

Kombinacija analize kolokacija i konkordancija, nakon pretprocesiranja teksta, omogućuje dublje istraživanje jezičnih karakteristika teksta unutar diskurzivnog konteksta. Ove tehnike pomažu u identifikaciji različitih obrasca, trendova i specifičnosti jezične upotrebe, pružajući uvid u šire značenje i funkciju teksta unutar određenog diskurzivnog okvira (Riahi i Sedghi, 2016).

5.1. Kolokacije

Prema Karlić i Bago (2021: 109) „kolokacije su niz ili kombinacija riječi čije je supojavljivanje češće od očekivanog slučajnog supojavljivanja“. Isto tako Seljan, Dunđer i Gašpar (2013) kolokacije definiraju kao nizove riječi ili pojmova koji se pojavljuju češće nego što bi se slučajno očekivalo. Ivir (1993, prema Stojić i Murica, 2010) pojam kolokacije definira kao:

> „susmještaj (lat. *com* „zajedno“ + *locare* „smjestiti“), odnosno suprotstavljanje ili kombiniranje riječi u sintagmatskome nizu. Riječi ulaze u kolokacije sa svojim prototipnim ili jezgrenim značenjem, koje se u kolokaciji oblikuje u konkretno, specificirano značenje koje favorizira ili dopušta konkretni kolokat.“

Drugim riječima, kolokacije su kombinacije riječi koje se često pojavljuju zajedno i imaju određenu semantičku vezu. Stojić i Murica (2010) navode kako je semantička uloga kod kolokacija čvršća i stroža nego u spoju slobodne sintagme jer zamjena jednog člana kolokacije rezultira novom kolokacijom posve novog značenja. Prema Seljan, Dunđer i Gašpar (2013), kolokacije se smatraju podskupom višerječnih izraza koji čine proizvoljne konvencionalne asocijacije riječi unutar određene sintaktičke strukture. Kao i kod drugih aspekata obrade prirodnog jezika, kontekst je vrlo važan (*GeeksforGeeks*, bez god.). U slučaju kolokacija, kontekst je dokument u obliku popisa riječi, a otkrivanje kolokacija u ovom popisu riječi znači pronalazak uobičajenih fraza (kolokata) koje se često pojavljuju u tekstu (*GeeksforGeeks*, bez god.). Ekstrakcijom kolokacija postiže se dublje razumijevanje dotičnog teksta, njegovog žanra, okruženja, stila, teme, tona itd. (Dunđer, Pavlovski i Seljan, 2020). Kolokacije su važne jer pružaju uvid u tipične jezične obrasce i konvencije u jeziku. Kolokati se raspoređuju u grupe definirane prema pravilima u jezično ovisnim gramatikama, a u svakoj grupi poredani su od najveće do najmanje vrijednosti (Karlić i Bago, 2021: 109).

Analiza kolokacija pomaže u razumijevanju upotrebe riječi u određenom kontekstu, razumijevanju konteksta i semantike riječi (Dunđer, Seljan i Odak, 2023), a osobito je korisna u leksikografskom istraživanju, računalnoj lingvistici, obradi prirodnog jezika i drugim područjima lingvistike (Karlić i Bago, 2021: 114). Obično uključuje korištenje statističkih metoda kako bi se odredilo koliko se često određene riječi pojavljuju zajedno i procijenilo koliko su njihove veze značajne. Kolokacije se mogu proučavati samostalno ili integrirano s okruženjem za vizualizaciju, u sustavima strojnog prevođenja ili u pronalaženju informacija (Seljan, Dunđer i Gašpar, 2013).

```
def analyze_collocations():
 ...
  # Pronalaženje Bigram kolokacija
    bigram_measures = BigramAssocMeasures()
    finder = BigramCollocationFinder.from_words(tokens)
    collocations = finder.score_ngrams(bigram_measures.pmi)
 ...
```

Programski kod 8. BigramCollocationFinder – analiza kolokacija

U programskom kodu 8 prikazana je implementacija analize kolokacija koja se provodi funkcijom `analyze_collocations()` na prethodno pretprocesiranom tekstu. Koristeći modul *BigramCollocationFinder* iz *nltk* biblioteke pronalaze se kolokacije, odnosno parovi

riječi koje se često pojavljuju zajedno u tekstu. U ovoj implementaciji koristi se PMI (eng. *Pointwise Mutual Information*) mjera asocijacije za određivanje i rangiranje kolokacija. PMI mjeri vjerojatnost da se dvije riječi pojave zajedno u tekstu, uzimajući u obzir njihove pojave u paru i pojedinačno.

Rezultat analize kolokacija je popis parova riječi koje se često pojavljuju zajedno u tekstu, zajedno s njihovim ocjenama kolokacije. Ove kolokacije mogu pružiti uvid u vezanost riječi u tekstu i pomoći u razumijevanju semantičkih veza između riječi.

5.2. Konkordancije

Postoje različite računalno potpomognute metode za analizu teksta, primjerice izrada konkordancija za proučavanje obrazaca upotrebe jezika (Dunđer i Pavlovski, 2018). Konkordancija je skup lokacija riječi zajedno s njihovim kontekstom (Kalpakchi i Boye, 2024). Često se koriste u lingvističkim istraživanjima kako bi se proučavala semantika, sintaksa ili upotreba određenih riječi ili izraza u stvarnom jezičnom korpusu (Dörpinghaus, 2024). Osim toga, analiza konkordancija primjenjiva je i na najstarije književne tekstove (Dunđer i Pavlovski, 2018) te se često koristi u analizi stila pisanja, prevoditeljstvu i leksikografiji.

> Konkordancija je „u širem smislu, usklađenost, podudaranje kakvih izjava, svjedočanstava, tekstova i sl. U užem smislu, potpun abecedni popis riječi i/ili pojmova, sa svim oblicima u kojima se javljaju, zajedno s minimalnim kontekstom i oznakom mjesta, koji se nalaze u nekom djelu ili u više djela (npr. istoga pisca, istoga žanra, istoga znanstvenog ili umjetničkog područja i sl.). Obično se izrađuju konkordancije najvažnijih djela (npr. Biblije) ili djela najvažnijih pisaca, npr. Shakespeareovih djela (usporedi na hrvatskome konkordancije nekih Marulićevih, Gundulićevih ili Krležinih djela). Konkordancije obično služe za produbljeniji studij ili tumačenje pojedinih djela te za raznolike analize (npr. lingvističke ili tekstološke)“ (*konkordancija*, Hrvatska enciklopedija).

Konkordancije su vrlo korisne u analizama koje zahtijevaju ispitivanje i razumijevanje konteksta u kojem se određena riječ, n-gram ili fraza koriste (Dunđer, Seljan i Odak, 2023). Na taj se način mogu uočiti različiti obrasci i trendovi u tekstu, kao što su učestalost ponavljanja određenog pojma ili fraze u različitim kontekstima, tvorba kolokacija riječi i dr. (Dunđer, Seljan i Odak, 2023). Konkordanciranje je proces ispisivanja riječi s njezinim kontekstima, ono omogućuje razumijevanje određene fraze s obzirom na kontekst u kojemu se pojavljuje (Dunđer, Pavlovski i Seljan, 2020). Dvojezične konkordancije se često upotrebljavaju u

strojnom prevođenju te predstavljaju proširenje rječnika, omogućujući pretraživanje jedinica od više riječi, kolokacija ili idiomatskih izraza, fraza ili čak cijelih rečenica (Jaworski, Dunđer i Seljan, 2021).

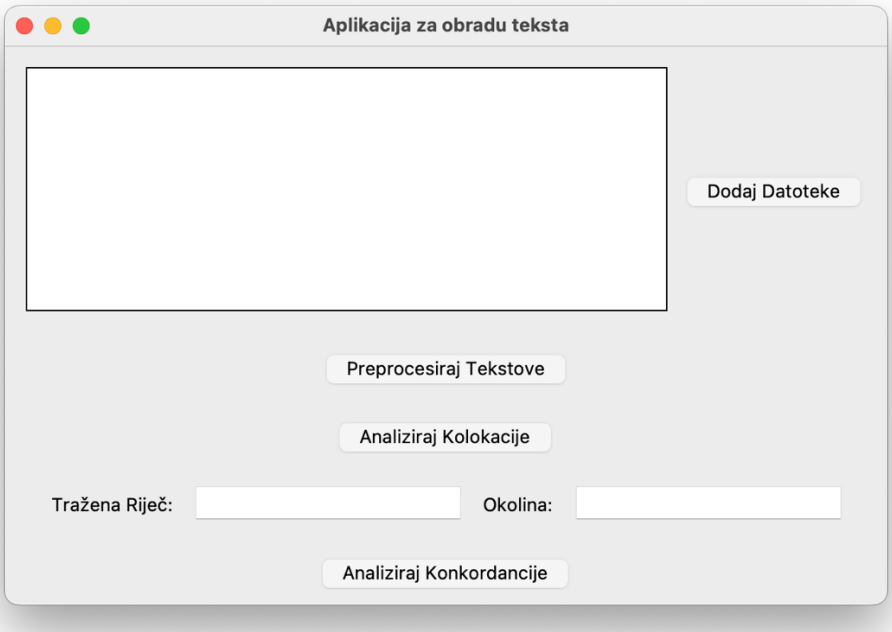
Analiza konkordancija je tehnika koja se koristi u analizi teksta za pronalaženje i prikazivanje svih pojavljivanja određene riječi ili fraze u korpusu teksta (Kalpakchi i Boye, 2024). Korisna je u mnogim područjima obrade prirodnog jezika, uključujući lingvistiku, leksikografiju, leksikologiju, te analizu teksta i korpusa, jer omogućuje istraživačima i analitičarima dublje razumijevanje upotrebe riječi ili fraza u različitim kontekstima, što može pružiti uvid u jezične obrasce, semantičke veze i druge lingvističke karakteristike jezika (Dörpinghaus, 2024). Analizom konkordancija omogućuje se dublje razumijevanje konteksta u kojem se određena riječ ili fraza pojavljuje (Dörpinghaus, 2024). Uz analizu konkordancija, u strojnom se prevođenju često upotrebljava tehnika pretraživanja konkordancija (eng. concordance searching) (Jaworski, Dunđer i Seljan, 2021). Pretraživanje konkordancija je tehnika traženja pojedinačnih riječi ili jedinica od više riječi iz prevedene rečenice u prijevodnoj memoriji, kako bi se potom pojavljivanja tih riječi prikazivala s odgovarajućim kontekstom (Jaworski, Dunđer i Seljan, 2021).

Analiza konkordancija u izrađenoj aplikaciji funkcionira tako da se unese riječ koju želimo analizirati (tražena riječ) i broj koji predstavlja koliko se riječi želi uključiti u okolinu tražene riječi. Na primjer, ako korisnik unese traženu riječ „kuća“ i okolinu 10, aplikacija će tražiti sve rečenice koje sadrže riječ „kuća“ te će uključiti 10 tokena prije i poslije te riječi u svakoj pronađenoj rečenici. Nakon unosa tražene riječi i okoline u polja unutar korisničkog sučelja (Slika 4), pritiskom na gumb „Analiziraj Konkordancije“, aplikacija iterira kroz prethodno pretprocesirani tekst i traži sve rečenice koje sadrže traženu riječ. Za svaku pronađenu rečenicu, aplikacija izdvaja okolinu tražene riječi i zatim ih sprema u novu datoteku. Ovaj proces omogućuje korisniku da brzo pronađe i pregleda rečenice u kojima se nalazi određena riječ, uz mogućnost uvida u kontekst koji okružuje tu riječ.

6. Korisničko sučelje

Korisničko sučelje ove aplikacije izrađeno je korištenjem Tkintera, standardnog grafičkog korisničkog sučelja (eng. *Graphical User Interface*, GUI) za Python. Tkinter je popularan alat za izradu desktop aplikacija zbog svoje jednostavnosti korištenja i širokoj dostupnosti. Omogućuje stvaranje prozora, gumba, okvira, ulaznih polja i drugih elemenata korisničkog sučelja.

Sučelje izrađene aplikacije sastoji se od glavnog prozora u kojem se nalazi okvir s popisom datoteka, gumb za dodavanje datoteka, gumb za pokretanje obrade teksta, analizu kolokacija te gumb i okviri za unos riječi i okoline konkordancija.



Slika 4. Snimka zaslona korisničkog sučelja aplikacije za obradu teksta

Glavni dijelovi sučelja su:

1. Tkinter prozor – stvara se pozivom funkcije `tk.Tk()` koja inicijalizira novi glavni prozor aplikacije.
2. Okvir za prikaz popisa datoteka – implementiran je pomoću `tk.Listbox` widgeta koji omogućuje prikaz više stavki u obliku popisa. Ovaj okvir prikazuje nazive datoteka koje su korisnici odabrali za obradu.

3. Gumb za dodavanje datoteka – omogućuje korisnicima odabir datoteka koje žele obraditi. Klikom na ovaj gumb, otvara se dijaloški prozor koji omogućuje odabir jedne ili više datoteka iz lokalnog sustava.
4. Gumb za pokretanje obrade teksta – klikom na gumb aktivira se funkcija koja učitava odabrane datoteke, provodi pretprocesiranje teksta te sprema rezultate obrade u novu datoteku.
5. Gumb za analizu kolokacija – klikom na gumb pokreće se funkcija kojom se provodi analiza kolokacija na prethodno pretprocesiranom tekstu. Rezultati se spremaju u novoj datoteci.
6. Polja za unos tražene riječi i okoline – omogućuju unos parametara potrebnih za analizu konkordancija.
7. Gumb za analizu konkordancija – nakon unosa tražene riječi i odabira okoline, klikom na gumb pokreće se proces pronalaženja svih pojavljivanja tražene riječi u tekstu zajedno s okolinom koja je odabrana. Rezultati se spremaju u novoj datoteci.
8. Dijaloški okviri za obavijesti – koriste se za obavještavanje korisnika o uspješnosti izvršenih zadataka ili o eventualnim problemima koji su se pojavili tijekom procesa obrade teksta.

Ovi dijelovi sučelja omogućuju korisnicima jednostavno i učinkovito upravljanje funkcionalnostima aplikacije za obradu teksta, pružajući im kontrolu nad procesom analize teksta i pregledom rezultata.

Kako bi korisničko sučelje bilo intuitivno i korisno, razmatrane su sljedeće karakteristike:

- Jednostavnost korištenja: jednostavno i intuitivno sučelje omogućuje jednostavan odabir datoteke i pokretanje obradu i analize.
- Preglednost: prikaz popisa datoteka omogućuje jasan pregled odabranih datoteka, također, jasno označeni gumbi olakšavaju izvršenje željenih radnji nad tekstom.
- Obavijesti i povratne informacije: upotreba dijaloških okvira za obavijesti i povratne informacije o statusu obrade ili o bilo kojim drugim važnim događajima.
- Prilagodljivost: mogućnost odabira više datoteka odjednom i fleksibilnost u odabiru vrste datoteka za obradu.

Kao rezultat, korisničko sučelje omogućava korisnicima učinkovito i jednostavno upravljanje procesom obrade tekstualnih datoteka, pružajući im potrebne alate i informacije za uspješno obavljanje željenih zadataka.

7. Testiranje i usporedba s drugim alatima

Testiranje aplikacije je važno kako bi se osiguralo da funkcije i metode korištene u aplikaciji ispravno rade te da proizvode očekivane rezultate. Standardna Python biblioteka ima ugrađene alate za podršku testiranju. Iako u Python ekosustavu postoji mnogo alata za testiranje, *Pytest* i *unittest* se ističu zbog svoje jednostavnosti korištenja i sposobnosti da se nose sa sve složenijim potrebama testiranja (*Dataquest*, 2022).

Jedinični testovi (eng. *Unit tests*) su segmenti koda napisani za testiranje drugih dijelova koda, obično jedne funkcije ili metode, koja se naziva jedinicom (eng. *unit*). Modul *unittest* radi na temelju objektu orijentiranih koncepata (*Dataquest*, 2022). Ovi su testovi vrlo važan dio procesa razvoja aplikacija, programa i softvera, jer pomažu osigurati da kod radi kako je predviđeno.

Testni slučaj (eng. *Test case*) se smatra jednom jedinicom testiranja i predstavlja ga klasa `TestCase` (*Dataquest*, 2022). Ova je klasa jedna od najvažnijih klasa za testiranje koda, a koristi se kao osnova za postavljanje testa, te omogućuje izvođenje više testova istovremeno (*Dataquest*, 2022).

Primjerice, testiranje funkcionalnosti pretprocesiranja teksta u izrađenoj aplikaciji uključuje provjeru ispravnosti pretprocesiranja datoteke koja sadrži tekst. Testiranje je provedeno u par koraka, kako je prikazano u programskom kodu 9:

1. priprema testnog okruženja – definiranje testnih datoteka koje će se koristiti za provjeru pretprocesiranja teksta;
2. pozivanje funkcije za pretprocesiranje (koja će provoditi pretprocesiranje teksta na testnim datotekama):
    a. Prvo se poziva funkcija `preprocess_pipeline()` s praznim tekstom kako bi se provjerilo je li povratna vrijednost funkcije tipa string. To se postiže pomoću naredbe `self.assertIsInstance` koja provjerava je li povratna vrijednost funkcije instanca klase string (*Dataquest*, 2022).
3. provjera ispravnosti rezultata pretprocesiranja – uključuje provjeru ispravnosti formata, sadržaja i ostalih karakteristika rezultirajućih datoteka;
4. ponovno pozivanje funkcije – ovaj put s određenim ulaznim tekstom `input_text`:
    a. definiranje očekivanih rezultata pretprocesiranja na temelju ulaznih podataka i očekivanog ponašanja funkcije. Očekivani rezultat nakon pretprocesiranja

teksta je `expected_output`. To se provjerava pomoću naredbe `self.assertEqual` koja uspoređuje povratnu vrijednost funkcije s očekivanim rezultatom (*Dataquest*, 2022).

Ovaj testni slučaj pomaže osigurati da funkcija radi ispravno i da se rezultati pretprocesiranja teksta podudaraju s očekivanjima. S obzirom na to da su oba testa prošla bez greške, može se zaključiti da je funkcija `preprocess_pipeline()` ispravno implementirana i da vraća očekivane rezultate.

```
import unittest
from apk_obrada_analiza import preprocess_pipeline

class TestTextProcessing(unittest.TestCase):

    def test_preprocess_pipeline(self):
        # Testiramo da li funkcija vraća string
        self.assertIsInstance(preprocess_pipeline(''),
str)

        # Testiramo da li funkcija vraća očekivani
rezultat
        input_text = "Ovo je neki tekst kojeg testiramo.
Ovo je drugi testni tekst."
        expected_output = "ovo je neki tekst
testiramo\novo je drugi testni tekst\n"
        self.assertEqual(preprocess_pipeline(input_text),
expected_output)
```

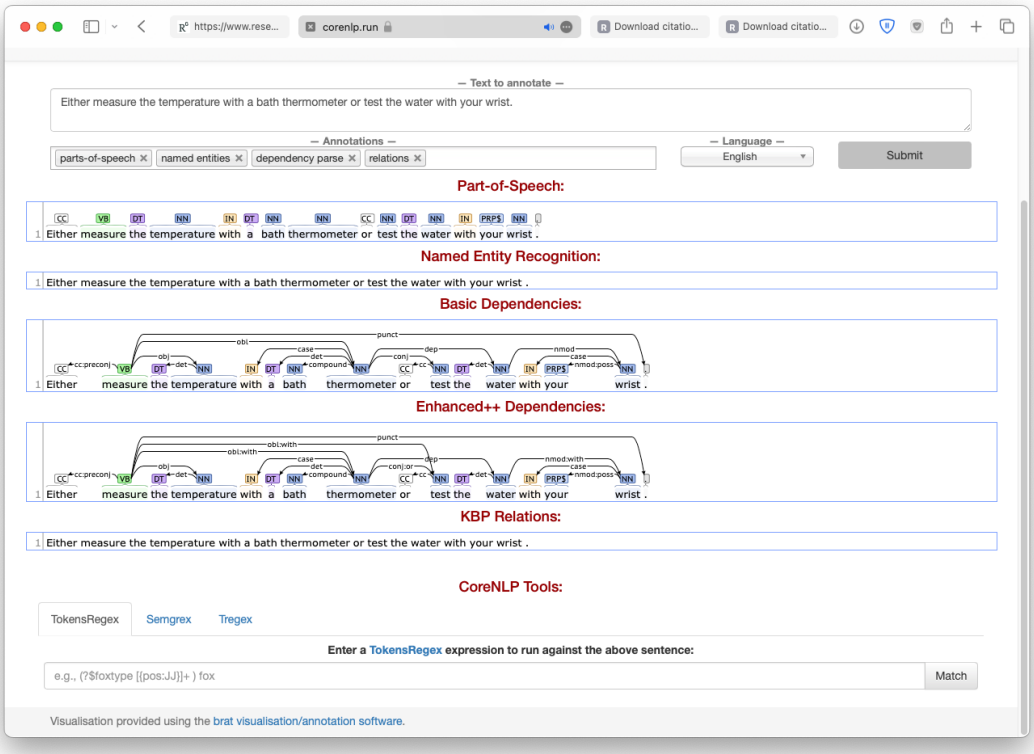
Programski kod 9. Testiranje pretprocesiranja teksta

Testiranje preostalih funkcionalnosti (za generiranje kolokacija i konkordancija) je također izvršeno u ovom diplomskom radu.

DRUGI ALATI ZA PRETPROCESIRANJE I OBRADU TEKSTA

Jedan od alata koji uspješno provodi pretprocesiranje i obradu teksta je *Stanford CoreNLP*. Ovaj paket alata omogućuje obradu prirodnog jezika na engleskom, arapskom, kineskom, francuskom, njemačkom i španjolskom jeziku, a razvila je grupa Stanford NLP. Uz pretprocesiranje, ovaj alat omogućuje funkcionalnosti poput: označavanja dijela govora (eng. *POS tagging*), prepoznavanje imenovanih entiteta, analizu sentimenta i parsiranje ovisnosti. CoreNLP je napisan u Javi i zahtijeva njenu instalaciju na računalu, ali također nudi

programska sučelja za nekoliko popularnih programskih jezika, uključujući i Python. Na slici 5 prikazana je online (demo) verzija alata.



Slika 5. Snimka zaslona – Stanford CoreNLP

Drugi uspješan softverski okvir otvorenog koda za obradu prirodnog jezika i analizu teksta je GATE (General Architecture for Text Engineering). Pruža niz alata i resursa za razne NLP zadatke, uključujući ekstrakciju informacija, rudarenje teksta, analizu sentimenta i semantičko označavanje. Može se koristiti kroz grafičko korisničko sučelje, a dodatno nudi razne API-je za programski pristup i integraciju u korisničke aplikacije.

Ipak, najviše je alata za pretprocesiranje teksta razvijeno za potrebe rudarenja teksta, a jedan od takvih alata je WordStat. Ovaj alat omogućuje značajke za pretprocesiranje teksta, analizu sadržaja, izdvajanje koncepata, analizu sentimenta i vizualizaciju rezultata. Sličan alat je RapidMiner, platforma za podatkovnu znanost koja nudi mogućnosti obrade teksta zajedno s drugim funkcijama rudarenja podataka i strojnog učenja. U RapidMineru moguće je kreirati tokove rada povlačenjem i ispuštanjem operatora za obradu teksta (npr. tokenizacija, ishodište, analiza sentimenta) i njihovim povezivanjem kako bi se definirao tijek podataka, nakon čega se pokreće proces za izvođenje specificiranih zadataka analize teksta.

8. Zaključak

Neobrađeni tekstualni podaci često sadrže šum, nepravilnosti i nedosljednosti, stoga je pretprocesiranje teksta ključno kako bi se pospješila učinkovitost NLP tehnika i algoritama. Čišćenjem i standardiziranjem teksta, osigurava se kvaliteta i pouzdanost podataka koji se koriste za analizu. Primjerice, uklanjanje interpunkcijskih i specijalnih znakova te zaustavnih riječi smanjuje šum i pojednostavljuje tekst, olakšavajući modelima „izvlačenje“ smislenih uvida. Za postizanje što boljih rezultata, različite vrste tekstualnih korpusa zahtijevaju različite metode pretprocesiranja, međutim, ne postoje univerzalne smjernice kako pretprocesiranje teksta primijeniti na potpuno novi tekstualni korpus. No, fleksibilnost u primjeni različitih tehnika pretprocesiranja omogućuje adekvatnu obradu tekstualnih podataka u skladu s ciljevima analize i karakteristikama korpusa teksta.

Pretprocesiranje teksta priprema „sirovi“ tekst za daljnju analizu, uključujući analizu kolokacija i konkordancija. Bez pretprocesiranja, analize ovog tipa mogu biti otežane zbog nečistoća u tekstu. Analize kolokacija i konkordancija pružaju dublji uvid u jezične obrasce i semantičke veze u tekstu, što može uvjetovati daljnje korake analize ili obrade teksta. Pretprocesiranje teksta često uključuje korake koji olakšavaju analizu kolokacija i konkordancija, poput tokenizacije i uklanjanja neželjenih znakova. Pretprocesiranje teksta također može imati značajan utjecaj na performanse i efikasnost algoritama strojnog učenja u obradi teksta. Dobro obavljeno pretprocesiranje može rezultirati smanjenjem dimenzionalnosti podataka, olakšavajući treniranje modela. Važno je istaknuti da pretprocesiranje teksta može biti prilagođeno specifičnim potrebama i ciljevima analize. Na primjer, u nekim slučajevima može biti potrebno provesti dodatne korake kao što su detekcija i zamjena sinonima ili ekstrakcija ključnih fraza radi dubljeg razumijevanja konteksta. Osim toga, pretprocesiranje teksta može biti posebno važno u situacijama kada se radi s tekstualnim podacima na više jezika. Različiti jezici mogu zahtijevati različite tehnike pretprocesiranja kako bi se adekvatno rukovalo njihovim specifičnostima i osigurala konzistentnost u analizi.

Pretprocesiranje teksta nije samo mehanički postupak; ono zahtijeva dublje razumijevanje jezičnih karakteristika i konteksta teksta kako bi se postigla visoka kvaliteta rezultata. Stoga, istraživanje i primjena naprednih tehnika pretprocesiranja, poput analize kolokacija i konkordancija, mogu dodatno poboljšati interpretaciju i relevantnost analize. Osim toga, pretprocesiranje teksta može biti ključno u kontekstu interpretabilnosti rezultata analize. Čišći, standardizirani tekstualni podaci mogu olakšati interpretaciju i razumijevanje rezultata analize.

Kontinuirano istraživanje i razvoj u području pretprocesiranja teksta doprinosi napretku u analizi i obradi teksta, otvarajući put za razvoj novih i inovativnih metoda i alata koji će omogućiti bolje razumijevanje i iskorištavanje bogatstva informacija sadržanih u tekstualnim podacima. U konačnici, uloga pretprocesiranja teksta u analizi i obradi teksta je neosporno važna i treba se smatrati ključnim korakom u svakom projektu koji uključuje rad s tekstualnim podacima. Ulaganje dovoljno vremena i resursa u pravilno pretprocesiranje može značajno poboljšati kvalitetu i pouzdanost rezultata analize te omogućiti dublje razumijevanje sadržaja teksta.

9. Literatura

1. Awan, A. (2023). *What is tokenization?* DataCamp. Pristupljeno 29.04.2024. s https://www.datacamp.com/blog/what-is-tokenization
2. Banks, G. C., Woznyj, H. M., Wesslen, R. S., Ross, R. L. (2018). A review of best practice recommendations for text analysis in R. *Journal of Business and Psychology*, 33(4), 445-459.
3. Bird, S., Klein, E., Loper, E. (2009). *Natural Language Processing with Python*. Sebastopol: O'Reilly Media Inc.
4. Chai, C. P. (2023). Comparison of text preprocessing methods. *Natural Language Engineering*, 29(3), 509–553.
5. Chowdhury, G. (2003). Natural language processing. *Annual Review of Information Science and Technology*, 37 (1). 51-89. Pristupljeno 20.03.2024. s https://strathprints.strath.ac.uk/2611/
6. Covington, D. (2016). Analytics : *Data science, data analysis, and predictive analytics for business*. Charleston: CreateSpace Independent Publishing Platform.
7. *Dataquest* (2022). *A Beginner's Guide to Unit Tests in Python*. Pristupljeno 04.05.2024. s https://www.dataquest.io/blog/unit-tests-python/
8. Das, M. (2023). *Python-docx: A Comprehensive Guide to Creating and Manipulating Word Documents in Python.* Pristupljeno 03.05.2024. s https://medium.com/@HeCanThink/python-docx-a-comprehensive-guide-to-creating-and-manipulating-word-documents-in-python-a765cf4b4cb9
9. Dean, J. (2014). *Big data, data mining and machine learning: value creation for business leaders and practitioners*. New Jersey: SAS Institute Inc.
10. Denny, M. J., Spirling, A. (2018). Text Preprocessing For Unsupervised Learning: Why It Matters, When It Misleads, And What To Do About It. *Political Analysis* 26(2), 168–189.
11. Dovedan Han, Z. (2021). *Progovorimo pythonovski.* Zagreb: Vlastita naklada.

    Dörpinghaus, J. (2024). Automated annotation of parallel bible corpora with cross-lingual semantic concordance. *Natural Language Engineering*, 1–24.

12. Dunđer, I., Pavlovski, M. (2018). Computational concordance analysis of fictional literary work. Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 0644-0648, doi: 10.23919/MIPRO.2018.8400121. Pristupljeno 30.05.2024. s https://www.researchgate.net/publication/326707667_Computational_concordance_analysis_of_fictional_literary_work

13. Dunđer, I., Pavlovski, M., Seljan, S. (2020). Computational Analysis of a Literary Work in the Context of Its Spatiality. In: Rocha, Á., Adeli, H., Reis, L., Costanzo, S., Orovic, I., Moreira, F. (eds) Trends and Innovations in Information Systems and Technologies. WorldCIST 2020. Advances in Intelligent Systems and Computing, vol 1159. Springer, Cham. https://doi.org/10.1007/978-3-030-45688-7_26. Pristupljeno 12.06.2024. s https://www.researchgate.net/publication/341450098_Computational_Analysis_of_a_Literary_Work_in_the_Context_of_Its_Spatiality

14. Dunđer, I., Seljan, S., Odak, M. (2023). Data Acquisition and Corpus Creation for Phishing Detection. Proceedings of the 46th MIPRO ICT and Electronics Convention (MIPRO), pp. 533-538, doi: 10.23919/MIPRO57284.2023.10159904. Pristupljeno 30.05.2024. s https://www.researchgate.net/publication/372024307_Data_Acquisition_and_Corpus_Creation_for_Phishing_Detection

15. Eldridge, S. (2024). *data analysis. Encyclopedia Britannica.* Pristupljeno 10.04.2024. s https://www.britannica.com/science/data-analysis

16. Friedl, J. E. F. (2006). *Mastering Regular Expressions.* Sebastopol: O'Reilly Media Inc.

17. GeeksforGeeks (bez god.). Text Preprocessing in Python. Pristupljeno 28.04.2024. s https://www.geeksforgeeks.org/text-preprocessing-in-python-set-1/

18. Golbeck, J., Robles, C., Edmondson, M., Turner, K. (2011). Predicting Personality from Twitter. *IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing*, *PASSAT/SocialCom 2011*. 149-156.

19. Hickman, L., Thapa, S., Tay, L., Cao, M., Srinivasan, P. (2020). Text Preprocessing for Text Mining in Organizational Research: Review and Recommendations. *Organizational Research Methods*, 25, 114-146.

20. Jaworski, R., Dunđer, I., Seljan, S. (2021). Usability Analysis of the Concordia Tool Applying Novel Concordance Searching. In: Rocha, Á., Ferrás, C., López-López, P.C.,

Guarda, T. (eds) Information Technology and Systems. ICITS 2021. Advances in Intelligent Systems and Computing, vol 1330. Springer, Cham. https://doi.org/10.1007/978-3-030-68285-9_14. Pristupljeno 15.06.2024. s https://www.researchgate.net/publication/348902428_Usability_Analysis_of_the_Concordia_Tool_Applying_Novel_Concordance_Searching

21. Kadhim, A. (2018). An Evaluation of Preprocessing Techniques for Text Classification. *International Journal of Computer Science and Information Security*, 16(6) str. 22-32. Pristupljeno 02.05.2024. s https://www.researchgate.net/profile/Ammar-Kadhim-4/publication/329339664_An_Evaluation_of_Preprocessing_Techniques_for_Text_Classification/links/5c1b6aa6a6fdccfc705ae648/An-Evaluation-of-Preprocessing-Techniques-for-Text-Classification.pdf
22. Kalpakchi, D., Boye, J. (2024). Quinductor: A multilingual data-driven method for generating reading-comprehension questions using Universal Dependencies. *Natural Language Engineering*, 30(2), 217–255.
23. Karlić, V., Bago, P. (2021). (Računalna) pragmatika. Temeljni pojmovi i korpusnopragmatičke analize. Zagreb: FF Press.
24. Kern, M. L., Park, G., Eichstaedt, J. C., Schwartz, H. A., Sap, M., Smith, L. K., Ungar, L. H. (2016). Gaining insights from social media language. *Psychological Methods*, 21(4), 507-525.
25. *klasa.* Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža. Pristupljeno 31.05.2024. s https://www.enciklopedija.hr/clanak/klasa
26. Kobayashi, V. B., Mol, S. T., Berkers, H. A., Kismiho, G., Den Hartog, D. N. (2018). Text mining in organizational research. *Organizational Research Methods*, 21(3), 733-765.
27. *konkordancija*. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža. Pristupljeno 31.05.2024. s https://www.enciklopedija.hr/clanak/konkordancija
28. Lo, R. T., He, B., Ounis, I. (2005). Automatically Building a Stopword List for an Information Retrieval System. J. Digit. Inf. Manag., 3, 3-8. Pristupljeno 30.05.2024. s http://terrierteam.dcs.gla.ac.uk/publications/rtlo_DIRpaper.pdf
29. McKinney, W. (2018). *Python for Data Analysis.* Sebastopol: O'Relly Media Inc.
30. Pavlovski, M., Dunđer, I. (2018). Is Big Brother Watching You? A Computational Analysis of Frequencies of Dystopian Terminology in George Orwell's 1984. Pristupljeno

11.06.2024. s. https://www.researchgate.net/publication/326704952_Is_big_brother_watching_you_A_computational_analysis_of_frequencies_of_Dystopian_Terminology_in_George_Orwell's_1984

31. Pennebaker, J. W., Boyd, R. L., Jordan, K., Blackburn, K. (2015). *The development and psychometric properties of LIWC2015*. University of Texas at Austin.
32. PyPi, (bez god.). *Python Package Index.* Pristupljeno 31.05.2024. s https://pypi.org/project/PyMuPDF/
33. Riahi, N., Sedghi, F. (2016) Improving the Collocation Extraction Method Using an Untagged Corpus for Persian Word Sense Disambiguation. *Journal of Computer and Communications*, 4, 109-124. Pristupljeno 31.05.2024. s https://www.scirp.org/pdf/JCC_2016042216265099.pdf
34. Sakar, D. (2016). *Text Analytics with Python: A Practical Real-World Approach to Gaining Actionable Insights from Your Data.* New York: Apress. Pristupljeno 27.03.2024. s https://download.e-bookshelf.de/download/0008/3870/64/L-G-0008387064-0017200370.pdf
35. Salton, G. (1988). *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Publishing Company, Inc.
36. Seljan, S., Dunđer, I., Gašpar, A. (2013) From Digitisation Process to Terminological Digital Resources. Proceedings of the 36th International Convention MIPRO 2013, 1053-1058. Pristupljeno 11.06.2024. s https://scholar.google.com/citations?view_op=view_citation&hl=en&user=POxqzeMAAAAJ&citation_for_view=POxqzeMAAAAJ:mVmsd5A6BfQC
37. Shlens, J. (2014). *Notes on Kullback-Leibler Divergence and Likelihood Theory*. Pristupljeno 01.05.2024. s https://arxiv.org/pdf/1404.2000
38. Srividhya, V., Anitha, R. (2010). Evaluating preprocessing techniques in text categorization. *International journal of computer science and application*, 47(11), 49-51.
39. Stojić, A., Murica, S. (2010). *Kolokacije – teorijska razmatranja i primjena u praksi...* FLUMINENSIA, 22 (2), 111-125. Pristupljeno 30.05.2024. s https://hrcak.srce.hr/file/97884

40. Šuman, S. (2021). *Pregled metoda obrade prirodnih jezika i strojnog prevođenja*. Zbornik Veleučilišta u Rijeci, 9(1), 371-384. https://doi.org/10.31784/zvr.9.1.23

41. Torres-Moreno J.-M. (2014). *Automatic Text Summarization*. Hoboken, NJ, USA: John Wiley & Sons.

42. Vijayarani, S., Ilamathi, M. J., Nithya, M. N. (2015). Preprocessing Techniques for Text Mining – An Overview. *International Journal of Computer Science & Communication Networks*, 5(1), 7-16. Pristupljeno 27.03.2024 s https://www.researchgate.net/publication/339529230_Preprocessing_Techniques_for_Text_Mining_-_An_Overview

Popis slika

Popis programskih kodova

Izgradnja aplikacije za pretprocesiranje teksta

Sažetak

Pretprocesiranje teksta važna je tehnika koja se koristi za čišćenje i transformiranje nestrukturiranih tekstualnih podatka i njihovu pripremu za daljnju analizu. Standardni postupak pretprocesiranja teksta uključuje uklanjanje interpunkcije, transformiranje velikih u mala slova, uklanjanje stop riječi i HTML oznaka, korjenovanje, tokenizaciju i lematizaciju. Cilj ovog diplomskog rada je izgradnja aplikacije koja će skratiti proces pretprocesiranja teksta i omogućiti njenim korisnicima bolji fokus na istraživanje i obradu prirodnog jezika. Rad se sastoji od teorijskog i praktičnog djela. Teorijski dio fokusiran je na važnost pretprocesiranja teksta i tehnike korištene za pretprocesiranje. U praktičnom djelu prikazana je metodologija izgradnje aplikacije u programskom jeziku Python. Zaključno je provedeno testiranje i evaluacija aplikacije.

Ključne riječi: pretprocesiranje teksta, čišćenje nestrukturiranih podataka, obrada prirodnog jezika, Python programiranje

Building a text preprocessing application

Summary

Text preprocessing is an important technique used to clean and transform unstructured text data and prepare it for further analysis. Standard text preprocessing includes punctuation removal, lowercasing, removing stop words and HTML tags, stemming, tokenization, and lemmatization. The goal of this master thesis is to build an application that will shorten the text preprocessing process and enable its users to focus on natural language research and processing. The thesis consists of a theoretical and a practical part. The theoretical part is focused on the importance of text preprocessing and the techniques used for preprocessing. The practical part consists of presenting the methodology of building the application in the Python programming language. Finally, testing and evaluation of the application was carried out.

Keywords: text preprocessing, unstructured data cleaning, natural language processing, Python programming